FINGERPRINTING BASED LOCALIZATION WITH LORA

USING DEEP LEARNING TECHNIQUES

A Project

Presented to the faculty of the Department of Computer Science

California State University, Sacramento

Submitted in partial satisfaction of

the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Jait Nitinkumar Purohit

FALL

2019

© 2019

Jait Nitinkumar Purohit

ALL RIGHTS RESERVED

FINGERPRINTING BASED LOCALIZATION WITH LORA

USING DEEP LEARNING TECHNIQUES

A Project

by

Jait Nitinkumar Purohit

Approved by:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, Committee Chair

Xuyu Wang, Ph.D.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, Second Reader

Xiaoyan Sun, Ph.D.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date

Student: Jait Nitinkumar Purohit

I certify that this student has met the requirements for format contained in the University format manual and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, Graduate Coordinator \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Jinsong Ouyang, Ph.D. Date

Department of Computer Science

Abstract

of

FINGERPRINTING BASED LOCALIZATION WITH LORA

USING DEEP LEARNING TECHNIQUES

by

Jait Nitinkumar Purohit

Fingerprinting Localization, a common technique used in indoor positioning uses short-range radio frequency and addresses problems with multi-path. Although, there are challenges in fingerprinting approaches like spatial ambiguity, long distances, low-bandwidth, scalability, cost and size constraints. Additionally, using BLE and Wi-Fi in indoor and outdoor environments has proven less efficient in comparison to Lora, considering location accuracy, RSSI stability, and packet-drop in line-of-sight and non-line-of-sight scenarios in GPS based applications.

This Master Project implements fingerprinting localization using Lora (Long Range) technology for an indoor environment using Deep Learning models and also compares and improves publicly available approach for an outdoor environment. In the offline phase of fingerprinting, 2D data at different locations have been collected, gathering RSSI values from the gateways at a fixed location. The online phase estimates the mean location error using Deep Learning models. The results from multi-layer Deep Learning Neural Network models like Artificial Neural Network (ANN), Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) are compared here. Additionally, we have implemented hyper-parameter tuning to improve our results by changing optimizer parameters like learning-rate, batch-size, epochs and tuning model parameters like the number of hidden units, the number of layers, activation functions, and optimizers. Analysis of different parameters of LSTM has proved that using hyper-tuning parameters is very important considering back-propagation and overfitting problems in a smaller dataset.

Indoor localization is carried out in a university building and data has been collected on the third floor. Indoor experiments using DL techniques achieve 1.2-2.0 [m] of mean distance error. Additionally, we have compared our deep-learning techniques for publicly available outdoor data-source collected from several Lorawan base-stations and Lora nodes from Antwerp city, Belgium. Interpolation techniques using de-noising auto-encoders have helped to interpolate outliers for this data. Our results have demonstrated that mean distance error of 191.52 [m] from LSTM has out-performed results from the KNN algorithm and other DL models as LSTM can forget, remember and then update the necessary information using memory cell.

The whole approach has been implemented in a python-based framework called TensorFlow. Google Colab has been used to train models as we compare our results from different hardware accelerators like GPU and TPU. Libraries like scikit-learn and Keras for implementing different classifiers, pandas and numpy for data-preprocessing and seaborn and matplotlib for data visualization have been used in this project. Hardware devices used here are Dragino Lora gateway kit, Arduino, GPS Lora shield and a DHT11 sensor.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, Committee Chair

Xuyu Wang, Ph.D.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date

# 

# DEDICATION

# *To my family*

Who always supported, believed and had faith in me.

# *To my Professor’s*

Who influenced and motivated me to face challenges and overcome them

# *To my friends*

Who kept me going in this school life and taught me never to give up.

# ACKNOWLEDGMENTS

I am thankful to Dr. Xuyu Wang, for giving me this opportunity to enhance my learnings and skills in the field of developing an IoT use-case and improving it using Machine Learning and Deep-Learning techniques. I acknowledge him for showing trust in me and allowing me to solve the challenging task related to fingerprinting localization. I believe, he has a significant impact on my master’s project because his research area in the field of IoT and Deep-Learning has also helped me to accrue more insightful knowledge along-with my previous work-experiences in the same field. He was very supportive and always gave valuable feedback during break and weekends. I would also like to appreciate Dr. Xiaoyan Sun, my second reader, for her efforts in discussing the project architecture and reviewing the report with her valuable feedback. I would also like to acknowledge Dr. Haiquan Chen for teaching me the detailed concepts of Deep Learning and AI, that has helped me building this project work. Additionally, Dr. Yuan Cheng has motivated me in the field of IoT as he was my mentor at CSU IoT Challenge, GE Predix in Spring 2018. Lastly, this project won’t have been possible without the constant support and motivation from my family, friends and faculty and staff members of the Department of Computer Science at California State University, Sacramento, for their continuous administrative support and guidance through my course curriculum.

TABLE OF CONTENTS page#

[DEDICATION viii](#_Toc531528795)

[ACKNOWLEDGMENTS ix](#_Toc531528799)

[LIST OF FIGURES xii](#_Toc531528800)

Chapters

1. [**INTRODUCTION** 1](#_Toc531528801)

[1.1 Importance of Fingerprinting Localization 1](#_Toc531528803)

[1.2 Problems of Localization in indoor and outdoor environment 2](#_Toc531528804)

1.3 Lora based localization techniques ……………………………………. 3

1.4 Purpose and Report Formation………………………………………….3

1. [**BACKGROUND** 3](#_Toc531528805)

[2.1 Various Localization Techniques 3](#_Toc531528807)

[2.2 LoRa Architecture 9](#_Toc531528808)

[2.3 Deep Neural Networks 15](#_Toc531528809)

2.4 Recurrent Neural Network ……………………………………………15

2.5 Long Short-Term Memory Network ……………………………….... 16

2.6. Convolutional Neural Network ……………………………………… 16

2.7 Importance of Hyperparameter Tuning ……………………………… 17

1. [**PROJECT REQUIREMENTS** 17](#_Toc531528810)

[3.1 Indoor Experimental Setup 19](#_Toc531528812)

[3.2 Dragino LoRa Gateway Setup with Arudino 19](#_Toc531528813)

1. [**PROJECT ARCHITECTURE AND TECHNOLOGIES USED** 23](#_Toc531528818)

[4.1 System Architecture 23](#_Toc531528820)

[4.2 Technology Used 26](#_Toc531528821)

1. [**PROJECT IMPLEMENTATION** 29](#_Toc531528822)

[5.1 Data Collection 29](#_Toc531528824)

[5.2 Data Preprocessing 32](#_Toc531528825)

[5.3 Data Analysis 35](#_Toc531528826)

[5.4 Fingerprinting Map Generation 37](#_Toc531528827)

[5.5 Interpolation using Denoising Autoencoders 43](#_Toc531528828)

[5.6 Analysis of Deep Learning Techniques for naïve approach 47](#_Toc531528829)

5.7 Analysis of Deep Learning Techniques for probability approach …….48

5.8 KNN comparison with Deep Learning results for outdoor localization .49

5.9 Analysis of Indoor Localization Results ……….……………………...49

5.10 Impact of different design parameters on LSTM…………………….... 50

1. [**CONCLUSION** 49](#_Toc531528830)

**7. FUTURE WORK** ………………………………………………………….50

[BIBLIOGRAPHY .……………………………………………………………...51](#_Toc531528834)

# LIST OF FIGURES

Figures Page

[4. 1: System Architecture for the PodWeb Application……………..……………... 23](#_Toc531528617)

[4. 2: Dapp, Web3, and Ethereum Interaction………………….....………………… 25](#_Toc531528618)

[5. 1: Modal for Login Screen……………………………………………………… 29](#_Toc531528062)

[5. 2: Smart Contract to Store User Account and Email…………………………… 31](#_Toc531528063)

[5. 3: Structure of User Collection in Mongo DB………………………………….. 31](#_Toc531528064)

[5. 4: Upload Podcast File………………………………………………………….. 32](#_Toc531528065)

[5. 5: Structure of Podcast Detail Collection in Mongo DB……………………...... 33](#_Toc531528066)

[5. 6: Mapping of File Hash Value and User Account Address……………………. 34](#_Toc531528067)

[5. 7: Display List of Published Podcasts…………………………………………... 35](#_Toc531528068)

[5. 8: Consumer of a Specific Podcast……………………………………………... 37](#_Toc531528069)

[5. 9: Sample Snippet of PodsToken Smart Contract………………………………. 39](#_Toc531528070)

[5. 10: Sample Snippet of PodsTokenSale Smart Contract……………………….... 41](#_Toc531528071)

[5. 11: Structure of Token Purchase Detail Collection…………………………….. 42](#_Toc531528072)

[5. 12: Display List of Latest Podcast……………………………………………... 43](#_Toc531528073)

[5. 13: Podcast Detail Component…………………………………………………. 45](#_Toc531528074)

[5. 14: Confirm Purchase from Consumer………………………………………… 46](#_Toc531528075)

[5. 15: Display List of Purchased Podcast…………………………………………. 47](#_Toc531528076)

# CHAPTER 1

# INTRODUCTION

FINGERPRINTING LOCALIZATION WITH LORA USING DEEP LEARNING TECHNIQUES – is an implementation to improve fingerprinting localization with lora technology using deep learning techniques, for indoor and outdoor environments. Localization is very important in GPS based applications and hence resolving and enhancing it becomes crucial in non-linear environments. Lora is a communication protocol used in our system as it is efficient in comparison to Wi-Fi and Bluetooth technologies in such applications. This system uses Deep Learning techniques to improve fingerprinting localization results instead of naïve approaches. Before digging deep into the understanding of the project let us know its importance, the existing problems and purpose of this project.

# **Importance of Fingerprinting Localization**

Fingerprinting Localization is a common and the most efficient approach used in complex indoor and outdoor positioning applications and does not require line-of-sight measurements. With the advent of GPS based applications in the field of edge of Internet of Things, positioning of edge-devices and gateways become extremely important in the context of achieving maximum computation and accurate results. In complex indoor infrastructure, fingerprinting is effective in estimating the accurate position of devices in the context to the optimal received signal strength. It is also productive in outdoor applications like car navigation, cell-phone tracking and edge-to-edge communication with cloud over multiple gateways where object’s location is necessary for services and communication. Moreover, fingerprinting localization saves hardware installation cost and effort because of its network structure.

Fingerprinting localization mainly consists of two main phases: offline i.e training phase and online i.e testing phase. The offline phase involves mapping of signal strength information based on Lora gateway RSS fingerprints into database. The online phase is the testing stage to estimate and determine object’s location based on mathematical modeling techniques like pattern matching, euclidean distance, machine learning and deep learning [1].

# **Problems of Localization in indoor and outdoor environment**

Localization in indoor and outdoor environments brings lot of challenges due to complex infrastructure layouts, signal instability and non-linear line-of-sight measurements. The non-linear relation between sensor nodes and target base-stations or gateways create challenges in positioning them in an accurate manner. Typical and complex infrastructure layouts also lead to RSSI signal instability due to factors like height, line-of-sight and non-line-of sight scenarios [2]. Due to wide fluctuations of GPS and Wi-Fi signals in indoor environments lead to an inefficient localization. Spatial ambiguity is a major problem where similar fingerprints are observed for distant location in comparison to current location [2]. In outdoor applications where location of the object is moving, corresponds to a smaller number of RSSI readings, which impacts the localization accuracy [2].

# **LoRa based Localization technique**

LoRa stands for Long Range, is a DNA for Internet of Things. LoRa is a spread spectrum modulation technique based on Chirp Spread Spectrum (CSS) technology. Millions of sensor devices are interconnected using LoRa. It is effectively used for long-range communcation purposes. The idea to choose LoRa over Wi-Fi and Bluetooth for fingerprinting localization is because of its efficieny and accuracy in indoor and outdoor environments. Lora has more stable signal strength in comparison to Wi-Fi and Bluetooth and outperforms them in terms of packet-drop in all line-of-sight and non-line-of-sight scenarios. It also shows higher logarithmic relation with distance in path-loss comparison with other localization techniques. Therefore, LoRa is low-power robust communication protocol used in GPS free environments, sending small packets of data over long distances.

# **Purpose and Report Formation**

The purpose of this project is to address fingerprinting localization problems in a complex indoor and outdoor environments and to design an efficient solution with LoRa technology to estimate location using deep learning techniques. This approach will provide an optimal location estimation in comparison to other mathematical approaches like pattern matching and euclidean distance. Until now we have discussed the problems which currently exists for the fingerprinting localization and basic overview of the LoRa technology used to overcome these problems using deep learning techniques.

The rest of the report is organized as follows. In Chapter 2, we will briefly discuss about key-concepts used in our project, like background related to various localization techniques, LoRa technology, deep learning techniques like artificial neural networks, convolutional neural networks, recurrent neural network and long-short term memory and the importance of hyper-parameter tuning to optimize our deep learning results. The next couple of chapters will discuss about project requirements and architecture followed by implementation details. The last few chapters will cover about conclusion and future work.

# CHAPTER 2

BACKGROUND

Nowadays, Blockchain technology is becoming incredibly popular. But what are blockchains? How does blockchain technology work and what problems do they solve? How are they used? Many people find it difficult to understand what is blockchain. In simple words, “Blockchain is a chain of blocks that contain information. This technique was initially described in 1991 by a group of researchers and was initially intended to timestamp digital documents so that it’s not possible to backdate them or to tamper with them.”[1] This technology was not widely used until Satoshi Nakamoto adapted this technique in 2009 for the creation of the digital cryptocurrency named Bitcoin.



# **Blockchain Concepts**

***The blockchain*** is a distributed ledger with very particular technological underpinning. However, there is a difference between distributed ledger technology and blockchain technology. In distributed ledger technology, database records are not governed by the central authority. Someone would suggest that’s apparently what blockchain is all about. Its data is distributed and owned by the individual or group and not handled by the single authority. However, there are two main properties which set blockchain apart from distributed ledger technology which are cryptographic signing and linking groups of records in the ledger.

These two properties of blockchain along with distributed ledger is such that it is challenging to change the data after it is recorded inside a blockchain. Blocks are added to the chain of blocks which is the part of the network. Each block consists of,

* Data,
* Hash of the block
* Hash of the previous block.

The type of application or use case of the blockchain determines the type of data stored inside each block. For example, in a cryptocurrency blockchain, data can be the details of the sender, receiver and the amount of cryptocurrency to be transferred.

Each block also has a unique hash value which identifies a block explicitly in a blockchain and in turn the data that block contains. When a new block of information is created, its hash value is calculated and then assigned to that block. But whenever any content (data) of the block changes, it will also cause hash to change. Hence hash values are essential in detecting changes made to blocks. So, in other words, if the hash value of the block varies, it implies that the content of the block has changed or updated as well. Each block also contains a hash of the previous block. This creates a chain of orderly blocks which are associated with its previous block. This property is what makes them more secure. So, in a blockchain, all blocks refer to its previous block using hash value. The very first block in the blockchain is called Genesis block. Since it is the initial block in the chain, it does not refer to any block.

Given that the attacker wants to tamper a block, it will also change the hash value of that block. And since the next block connected to it will have a different hash value than that of the tampered block. The tampered block will not be attached to the chain as it will not have the value of its previous block (tampered) and will be considered objectionable to the whole blockchain [2]. So, changing the value on one block makes all other blocks after it invalid which in turn helps in preventing any sorts of tampering in the chain. For example, in Bitcoins, the required POW calculation time and associating that block to the chain takes around 10 minutes. Tampering with one block means that POW needs to be recalculated for all the remaining blocks after it. Because of this, it becomes challenging to tamper any block in a blockchain. Whole security of this system depends on this POW and hashing techniques. However, with high computational and processing capabilities, one can effectively tamper a block and then recalculate the hash values of all other blocks after that to maintain the flow, consistency, and validity of the blockchain system. Blockchain uses consensus algorithms to select a leader who will decide the contents of the next block. The two popular consensus algorithms are,

* Proof-Of-Work
* Proof-Of-Stake

Let first discuss ***Proof-Of-Work algorithm abbreviated as PoW.*** It is one of the most popular algorithms currently used for Bitcoin and Ethereum blockchain technology. To select the leader who would choose the next block to be added to the blockchain, one needs to search a solution for a complex mathematical problem. All the participant would compete to solve a complex issue. The hash function used is cryptographically secure, so brute force is the only technique to address this complex problem. All the participant’s involvement in solving the problem are known as miners. The success of this algorithm is mainly because of two reasons, difficulty in finding a solution and easy to verify the answer. A Miner who finds a solution is awarded some amount of currency and thus remain motivated to keep mining and keep earning. This is called as block reward which is credited in miner’s account and also known as transaction fees which are deducted from the person who initiated the transaction. Due to limited computational power, miners won’t think to approve the fraudulent transaction and waste the power without earning a currency. PoW provides the needed security. However, it is very energy and power consuming methodology. “With fewer miners than required mining for coins, the network becomes more vulnerable to a 51% attack. A 51% attack is when a miner or mining pool controls 51% of the computational power of the network and creates fraudulent blocks of transactions for himself while invalidating the transactions of others in the network.”[3] The miner having more computational power are more likely to solve the mathematical problem and earn rewards. A similar analogy is used in Proof-Of-Stake consensus algorithm.

Proof-Of-Stake abbreviated as PoS is the algorithm where there are no miners, people involved are known as validators. The selection of the leader from the set of validators is based on the amount of currency kept at stake to validate the transaction, add a transaction to the block and get rewarded for the same. Is there any advantage of using PoS over PoW? The answer to these questions are yes; there is definitely an advantage of using PoS over PoW, which is PoS takes away all the computing power and energy. The PoS seeks to address the issue caused using PoW by attributing mining power to the proportion of coins held by a validator. This way, we eliminate the need for utilizing energy to answer a mathematical puzzle, a PoS validator is limited to validate a percentage of transactions that is reflective of his or her ownership stake. For instance, a validator who owns 1% of the ether available can theoretically mine only 3% of the blocks. The person who holds more stake or more currency in the over-all network could mine more blocks eventually increasing the capital.

“With a PoS, the attacker would need to obtain 51% of the cryptocurrency to carry out a 51% attack. The proof of stake avoids this ‘tragedy’ by making it disadvantageous for a miner with a 51% stake in a cryptocurrency to attack the network. Although it would be difficult and expensive to accumulate 51% of a reputable digital coin, a miner with 51% stake in the coin would not have it in his best interest to attack a network which he holds a majority share. If the value of the cryptocurrency falls, this means that the value of his holdings would also fall, and so the majority stake owner would be more incentivized to maintain a secure network.”[3]

With recent developments in technology and its increasing demand in cryptocurrencies, Smart Contracts has evolved to exchange coins based on certain conditions automatically. Smart contracts are short programs stored on the blockchain. There are three types of blockchain,

* Public Blockchain
* Private Blockchain
* Consortium Blockchain

***Public Blockchain*** is open sourced, available to all and anyone can participate. Most of the public blockchain are based on Proof-Of-Work. Data is transparent but anonymous. They are fully decentralized. Anyone can access the data in this blockchain once those blocks are verified. Advantages for public blockchain are there is no infrastructural cost, secured as all nodes verify transactions and transparency among all the nodes. However, there are some drawbacks which there are is no privacy, inefficient as all the nodes verify all the transactions given that those all are miners. The best example for public blockchain is Bitcoin, Ethereum and Litecoin.

***Private Blockchain*** is permissioned blockchain. All permissions for this blockchain are kept centralized to one entity. They are designed to be verified internally by a restricted group of personnel only. These systems are more venerable to security breaches like any other centralized system. Advantages of private blockchain complete data privacy regulations, reduced data redundancies, no semi compliance mechanisms, efficient as verification is done by the owner only, the owner has control over reading/write privileges. However, there are some drawbacks such as controlling power is consolidated to a single entity, difficult for inter-organizational blockchain compatibility. Known examples for private blockchain are MONAX and Multichain. Its applications are related to the internal functions of the system like database management or auditing of any enterprise.

***Consortium blockchain*** is not entirely public. It lies between public and private. Some preselected nodes control it. So, for any block to be approved, it requires the majority of these nodes to make that block valid or accepted. They are not fully decentralized. Advantages of consortium blockchain are more privacy compared to public blockchains, higher in scalability, relatively fewer nodes to verify transactions. No controlling power of consolidation. Some of the known example for consortium blockchain are R3, B3i, and Corda.

# **Ethereum**

Ethereum is a distributed platform that utilizes the blockchain technology first used by Bitcoin in 2008 by Satoshi Nakamoto. It could be used to create a decentralized application using the blockchain technology. The use cases of Ethereum Blockchain is far beyond cryptocurrency. Vitalik Buterin first proposed it in late 2013 where he states the intention of Ethereum being to provide, “a blockchain with a built-in fully-fledged Turing-complete programming language that can be used to create "contracts" that can be used to encode arbitrary state transition functions.”[4] These qualities will give us the opportunity to use Ethereum blockchain as the backend for new decentralized and distributed applications, such as the one developed for this project. Let’s discuss a few basic concepts regarding Ethereum. Ether is the cryptocurrency used to trade value for specific commodities or asset.

***Smart Contract*** is a computer program which executes when certain conditions in the code successfully matched. It could be used to transfer and receive anything which has some value. It could be either money, tokens, content, information, any form of data which could consider having some value. The smart contract is stored on the blockchain, and so they are supposed to be immutable and distributed. Once an agreement has been created and deployed no one could tamper everyone on the network validates the code of the agreement and output of the agreement. So, tampering with smart contract or agreement is almost impossible for the person being a part of the system. Events could be written in the smart contracts which could be triggered whenever the people using the smart contract takes specific actions. There are many different use-cases where the smart contract could be used, but few examples are using smart contract in crowdfunding application, PodWeb application (current project), Banking applications to issue loans, election voting, insurance companies to issue claims and many more.

***Ethereum Virtual Machine*** abbreviated as EVM is the heart of the Ethereum blockchain technology. It eliminates the need of building the blockchain application from scratch. It handles all the connections and communication over millions of computer or nodes in the network. EVM is responsible for running and maintaining a smart contract. EVM is secured using the firewalls, it also eliminates the issues present in the system and reinforces security measures, which not only ensure that the platform is safe but also its tokens and transactions on the network stay secure and encrypted. EVM is the standalone individual layer isolated from the system. Because of this isolation, EVM is impervious of getting hacked or corrupted. A transaction that takes place using smart contract is executed on EVM, and it keeps track of change of state of blockchain network. EVM operates by charging some amount of ‘gas’ known as the fuel to change the current system state by executing a transaction. If ‘gas’ required by the EVM is less than the ‘gas’ provided while deploying the contract then, the transaction will be failed. If ‘gas’ provided while deploying the contract is greater than the amount of ‘gas’ required to execute the transaction, the remaining amount of gas is returned back to the user in the form of ethers who deployed the smart contract in the blockchain network. Smart contract once compiled using EVM it converts to the bytecode. Smart contracts are written using Solidity and Serpent programming languages. I have used the Solidity programming language for this project to write the smart contract. Once the contract gets compiled, we could store the bytecode and invoke the methods in the smart contract which will be referenced via bytecode.

***Solidity*** is one of the programming languages for writing a smart contract and is very similar to the JavaScript and follows object-oriented programming constructs. Solidity code runs on the Ethereum Virtual Machine. The extension for this file ‘.sol.’ And it is statically typed language, which means all the type will be checked before we ran the program. It also supports inheritance and complex user-defined types. Contracts in Solidity are like java classes. The code is compiled to the Ethereum Virtual Machine. Once it is deployed, it is completely isolated and cannot be reached outside of the EVM. So, a developer has no control over the code of contract once it is deployed. It works just like physically written contracts which can be created by anyone but manipulated by no one once it is signed. It is quite easy to write smart contracts in Solidity. We could define a struct for the custom defined datatype, along with built-in data types like uintX, where X belongs to 8,16,24,32 ... 256. It also supports string, bytes, array, and address as datatypes. Solidity also has a mapping of the key-value pair which could be used to store the data in the blockchain. There are many concepts for solidity, out of which the central idea which I needed to understand to make progress in my project was about the function types. Function types are merely the types of functions. “Variables of function type can be assigned from functions and function parameters of function type can be used to pass functions to and return functions from function calls.” [5] There are two types of function types,

* Internal: This type of function could only be called inside the current contract, and it cannot be invoked or executed outside of the context of the current contract.
* External: This type of function can be passed as parameter and returned whenever a function is invoked externally. It has an address and function signature.

Example of function is showed here,

***function (<parameter types>) {internal|external} [pure|view|payable] [returns (<return types>)]*** [5]

If the function is ***payable***, it means function except for 0 tokens or ethers. If the function is ***non-payable*** and if we send tokens or ethers to that function it will reject the request and give us unhandled promise exception.

If a function has a ***view*** defined in the function type, that means we would be able to read as well as write data using that functions.

If a function has ***purely*** defined in the function type, means we won’t be able to read the data, but we could use that function to write data. It might be useful for internal helpers’ functions and utilities relating to tasks like calculation, permission, and typecasting.

***The lifecycle of Transactions in Ethereum*** will be discussed in this section. Almost everything which interacts with the blockchain in Ethereum is considered as the transaction. The ***first stage*** in the lifecycle of the transaction is to create a raw transaction. Raw transactions are the way by which we segregate the procedure of creating, signing and deploying the raw transaction to the blockchain. A raw transaction is a JSON object which contains specific information. Keys in JSON object for the raw transaction are,

* Nonce: total transaction count for an account which is creating a transaction.
* gasPrice: Price per unit of the gas user is willing to pay for this transaction.
* gasLimit: Maximum gas you are willing to pay for this transaction.
* to: address to which we want to send this transaction, mostly a contract address or could be another user account address depending on use-case
* value: number of tokens or ether user wants to send.
* data: generate the hash of the function which needs to be invoked in the smart contract and hash of the argument which needs to be passed. Combination of these two hashes’ gives us the data payload.

The ***second stage*** of the transaction lifecycle is to sign the transaction. We could sign the raw transaction object using the private key of the user who is initiating the transaction. The ***third stage*** is validating the transaction locally. The signed transaction is validated locally to make sure that the given account address actually signed the signed transaction. The ***fourth stage*** is broadcasting of the signed and locally validated transaction to the network. The ***fifth stage*** miner nodes accept the transaction and store in the mining pool. Mining pool contains transactions with the gas price which will be awarded who solved the mathematical puzzle. In the ***sixth stage***, the miner who solve a mathematical problem and validates the transaction, add the transaction to the valid block. This valid block is agreed among all the nodes in the network and added to the chain of blocks.

There are two types of account in the Ethereum, Externally owned accounts and contract accounts. Externally owned accounts abbreviated as EOA’s has ether balance and are controlled using the unique private key for each account, and it has no code associated with it. However, Contract accounts abbreviated as CA’s have ether and token balance as well as associated code. The code is executed whenever a transaction is initiated, or event is triggered. Contract accounts can have their own permanent state and could call other contracts based on the instructions present in the code. The externally owned accounts initiate the transaction. Whenever a contract account receives a transaction, its code is executed. ‘Contract code is executed by the EVM on each node participating in the network as part of their verification of new blocks.’ Though the contracts cannot initiate their own transactions, they can send “messages” to other contracts. Messages are almost the same as transactions but are only initiated by CAs through their contract code, not by EOA’s. Like transactions, “messages” include the message sender, the message recipient, the ether amount to transfer, and an optional data field to call a function in the recipient's contract code.

Bitcoin blockchain is first fully decentralized blockchain which mainly focuses on the currency which could be used by millions of people. It is open-source and peer-to-peer electronic cash system which could replace the existing banking system. However, Ethereum is a programmable blockchain-based software platform where people could focus more on building decentralized applications. Currently, both Bitcoin and Ethereum use the same consensus protocol that is, Proof-Of-Work. Block time for Ethereum averages presently around 14 seconds while Bitcoin is about 10 minutes.

I have used Ethereum for this project, as it is Turing complete, it has good community support. Ethereum developers are continually providing support and enhancing the features of the Ethereum Blockchain. Ethereum is moving towards Proof-Of-Stake consensus protocol which would increase the evaluation of ether way more than bitcoin. I believe for my project, having the backbone of Ethereum blockchain would make it easy for the podcast consumers and publisher to transfer the tokens. All the metadata useful to maintain transparency and authenticity is stored in the blockchain using smart contract.

# **Interplanetary File System**

It is abbreviated as IPFS. It is a distributed file system which uses peer-to-peer protocol and each node stores a collection of the hashed files and directories. In simple terms, a client who wants to push the file to IPFS invokes an ‘add’ API call with the IPFS node installed locally or remotely on the cloud. This will give us hashed value for the file. To retrieve a file from IPFS, the client only needs to call the hash of the file it wants. IPFS then processes through the nodes and supplies the client with the file. There is a security issue with using this file system. Anyone who knows the hash value of the file could have access to the file. For the scope of this application, we have not used encryption and decryption for the hash value of the file, but in a real-world scenario, we should encrypt the hash value of the file. The latency of the blockchain network will increase more and more as we keep storing files. So, it is not advisable to save files on the blockchain network. Therefore, IPFS is powerful when utilized with blockchain. We keep simple data as part of the blockchain such as the hash value of the file, and we store the actual file on IPFS which gives us an excellent platform for file storage and decentralized peer-to-peer properties of the IPFS.

# CHAPTER 3

PROJECT REQUIREMENTS

The purpose of the PodWeb application is to provide the ability to the user with the way to sell, purchase and publish a podcast. The user could access the podcast on any platform Windows, MAC, Android or iOS by visiting a web application for this project. The user of the app won’t be aware of the user Ethereum address which will be generated while user sign-up. When a new user signs up, the user will be awarded a joining bonus in the form of ‘Pods’ token.

‘Pods’ token is ERC20 token created based on the Ethers. If the valuation of Ethers goes up or down, the value of Pods token could also go up or down respectively. ERC20 is standardly used for smart contracts on Ethereum blockchain for implementing tokens. ERC stands for Ethereum Request for comments, and 20 is the number assigned to this request. ERC20 defines a standard list of rules for Ethereum tokens to follow within the larger Ethereum ecosystem, allowing developers to predict the interaction between tokens accurately.

The data stored on the Ethereum blockchain is user account address, user email address, the path of the podcast file in the hash value, the address of the podcast file in hash value and if it is paid podcast then amount to purchase that paid podcast otherwise value will be 0. The podcast file itself will not be stored on the blockchain, because it would be expensive if we save files in the blockchain. Podcast files will be stored in the distributed file system. The path to the file and address of the file will be stored on the blockchain network. Users could either publish or purchase a podcast. So, there are two roles for any given specific users ‘publisher’ and ‘consumer.’ ‘Publisher’ is the user who publishes the podcast to the portal by uploading the files and setting its metadata, while ‘consumer’ is the keyword used whenever a user listens to the paid or free podcast.

If the user wants to publish a podcast, the user should upload a podcast file. For the scope of the project, only mp3 files are allowed. Uploading of files other than mp3 will give an error. However, in real-world application type of podcast file could be an audio/ video so publisher should be given flexibility of which file to be uploaded. These files are stored in the IPFS.

The motivation for the user to use this application is, the time taken to publish a podcast is less compared to the time taken by third-party platforms like iTunes and Google Play. And, full profit belongs to the publisher of the podcast. Publisher doesn’t have to give away any amount of the share to the third-party platforms. These features are enabled using smart contracts. The user interface for PodWeb tries to cover all the functionality as described below,

* Interface for the user to sign-up/login/steps to use application
* Interface to view and play the latest podcast arranged chronologically published by all users.
* Interface to view and play published podcast and form to upload a podcast
* Interface to display and play purchased podcast.
* Interface to purchase the token utilized to buy the podcast
* Interface to display and update the user details.
* Interface to update a password when a user forgets the password.



# **User Account Login/Signup**

Users of the PodWeb application when signs-up, a new user account address will be generated. This user account address is created using an API call made with a web3.js library which is used to connect to the blockchain network. Whenever a new user account address is generated, ten pods tokens are given to the newly joined user as the joining bonus which could be utilized in purchasing a podcast. This transaction of transferring ten pods token from an administrator account to the user account is recorded in the blockchain. Ethereum blockchain keeps track of the number of tokens held by a specific user account. It also holds which email address is associated to which user account address.

When a user tries to log in, along with email address and password, user account address stored in Ethereum blockchain network is also validated for a given email address to allow the user to log in successfully.

# **Reset Password**

Users will have the ability to reset the password. An email notification will be triggered with a unique identification number which will expire within 30 minutes of the email got triggered. The user will have to enter a new password and confirm the password to update the password. On the server end, the unique identification number will be decoded which would contain user account address, user email address and timestamp when this unique identification number is generated to validate if identification is expired or not.

# **Purchase Tokens**

Publisher or consumer of the podcast will have the ability to purchase the tokens. These tokens are ERC20 tokens deployed by the administrator in the form of a smart contract. The user will select some tokens which user wants to purchase, and appropriate conversion will take place in the backend to fill the amount in dollars which the user needs to pay to buy the token. The user could also add an optional note for future reference of why user bought these tokens. The amount for the selected tokens is decided based on the current ‘ether’ price. For this project, the price for the token is considered as one token would cost 0.001 ether. So, for example, the price of ether is 210 dollars, to purchase 50 tokens it would cost 0.05 ethers or approximately around 11 dollars. For the scope of this project, users will pay only dollars as user account address is not exposed to the users via a web application. For an administrator to accept this amount in dollars, stripe payment API has been integrated; if payment is successful, tokens will be transferred to the user from administrator to user account. This might take time based on the time taken for the transfer token transaction to get validated by the miners. Once, Ethereum transaction is successful; users will be able to view in tokens history of how many tokens user purchased and what is the transaction number, block number, current block hash where the transaction is placed and the timestamp of the transaction.

# **Upload Podcast**

For the scope of this application, an only mp3 file format for the podcast is supported. When a user uploads a podcast, the file format is validated. The user has to mention some metadata for the file to upload it for the other users to use. These metadata fields are, the title of the podcast, artist name of the podcast, tags related to the podcast, if the user decides to keep it as paid podcast then the user needs to specify the amount or else just select it as free podcast and amount will set to 0 tokens for the specific podcast. Client and server-side validation to upload metadata and file is performed. If validation is passed, the podcast file will be uploaded to the distributed file system. All the metadata for the podcast except amount and file path is stored in the database. User account address, the amount for the podcast and file path address is stored on the blockchain. Users will also have the ability to update the metadata about the file, but users are not given provision to update file for a specific podcast along with the cost to purchase particular podcast. A user who published or uploaded a podcast would also be able to see a list of all the podcast uploaded by oneself. Not only that, but the publisher of the podcast would also be able to see purchase history if it is a paid podcast.

# **Purchase Podcast**

The consumer of the podcast can purchase the podcast using the ‘Pods’ token used for this application. The consumer won’t be able to play podcast until they bought it. The source of the paid podcast will only be attached once user pays for the specific podcast. A number of tokens selected by the user will be validated when the user confirms to pay for a podcast. There are two validation checkpoints in this functionality. Firstly, sever will check if the amount for the podcast which user wants to purchase has been altered or not, if the amount is not changed we will have another validation check which is a number of tokens user holds in the network is greater than or equal to the number of tokens required to purchase a podcast. If all the validation checks are passed source for the podcast will be attached when the transaction is completed successfully. The user will also be able to see the list of all purchased podcast and have the ability to remove the podcast if not required. However, deleting a purchased podcast cannot be undone, and consumer needs to buy it again. The consumer of the purchased podcast can write comments for a podcast which could help another consumer to consider the option of purchasing. All users can like a podcast. Token history will be updated whenever user purchases a specific podcast with the details including transaction address, block number, block address, and timestamp.

# **Update User Information**

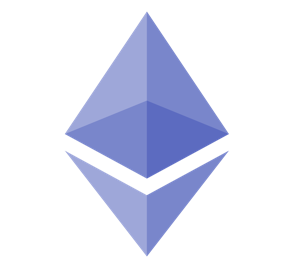
Registered user will be able to view the user details and update it if required. User metadata is stored in the database. It would be used for analytics purpose for future use. Users will be able to view a number of tokens held by oneself in the application and primary email address using which user registered in the application and both fields are not editable. Users will have the ability to update secondary email address, home address, city, state, and country information.

# CHAPTER 4

PROJECT ARCHITECTURE AND TECHNOLOGIES USED



# **System Architecture**

Client Ethereum Network

Controller

 **** 

Payment API IPFS Azure Cosmos DB

Figure 4. 1: System Architecture for the PodWeb Application

The underlying architecture of the PodWeb application is shown in Figure 4.1, focusing on the highest-level components. Podcast, user metadata are stored in the Azure Cosmos DB. All the interactions between publisher and consumer of the paid podcast are made it possible using the smart contract which is written for PodWeb application that runs on the Ethereum network. Administrator of the system deploys a smart contract to create one’s own cryptocurrency named as ‘Pods’ Token.

The controller plays a significant role as shown in figure 4.1. It interacts with not only the client but also Ethereum network, payment API, IPFS and database. The controller is the web service which handles communication, data passing among different components of the project. It allows us to set which Ethereum network to connect, select the appropriate database to store and manipulate the data, connects to the client to give excellent user experience for the end users. Data flows two-ways from the controller to the client. Whenever a consumer request to fetch user information or to purchase a podcast controller receives the request, processes it and send the response back with an appropriate message. Whenever, a client sends a request to obtain a podcast, operations, as shown in figure 4.2, will take place. Web services are written in Javascript which in turns interact with the blockchain via the web3 library. This web3 library is used to connect to a specific Ethereum network. For both upload and purchase web service calls, the parameters passed include the information needed to validate, that the file request exists, and the file's metadata can be found and that the file’s hash value found in the request exists in the blockchain.

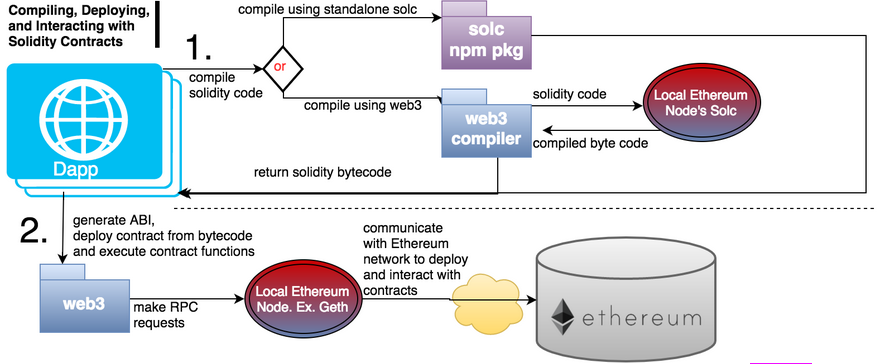


Figure 4. 2: Dapp, Web3, and Ethereum Interaction

Image credits: [Karl Floersch](https://karl.tech/5-essential-ethereum-dapp-tools/)

The controller also interacts with payment API. Payment API is used to receive the amount in dollars from the users who purchase the ERC20 token. Users card information is passed securely to the web service which handles interaction with the payment API. Payment gateway validates the user card information and when REST API call is made from controller to the payment gateway API. If payment gateway successfully authenticates the user's card details, it sends a success response to the controller which in turn allows the controller to decide to transfer the tokens to the user who posted a request. There is an interaction between a controller and IPFS. Whenever a user uploads a file, a web service call is made to the controller from the client along with metadata and file which needs to be uploaded. Before adding this podcast file to IPFS, the controller establishes a connection with the IPFS using an ‘IPFS-API’ library used for NodeJS. This library enables the controller to make a REST API call to the IPFS connection to add and get hash value of the file.

# **Technology Used**

For this project, I have used ***AngularJS*** to build a client-side portal for the PodWeb Application. Angular is a single-page application, and it is a prevalent, dynamic and very easily adaptable framework. Angular give us the platform to write a component either in Typescript or JavaScript and serves the pipeline which connects all the different components. It uses simple HTML; we don’t have to worry about program flows and what data should be loaded first. We must define HTML and components and rest everything is handled by the Angular behind the scenes. Angular also has data models which are POJO class, and it doesn’t require getter and setter methods. We could directly get and set the values from the variable itself. It allows us to use directives which will eventually bind together all the components or integrate one component with another component. Data-binding in angular means you don’t have to put data into the view manually. There are filters which allow us to manipulate data on the view level without changing our controller’s code. Angular also support context-aware communication between two or more components. ‘Broadcast’ in angular will send a message to all children components, while ‘emit’ will send a message to all ancestor’s components. We have already covered the majority of Ethereum topics and why I have used Ethereum for this project along with IPFS in previous chapters. So, let's discuss Azure Cosmos DB.

In this project, I have used ***Azure Cosmos DB***; it is a globally distributed multi-model database service for building common scalable applications. It is an agnostic schema database that automatically indexes all the data providing high-speed performance. It allows users to make a data model with the help of the API. Azure contains many different types of data models. The user can create data in the cloud in various regions of the world. It is scalable with guaranteed low latency. The user has the option to select from multiple available models like the graph API model, document DB model, key/value model or column family model. This service provides a modern way of developing apps with multiple language compatibility. MongoDB stores data in JSON style document format which has a versatile structure for the dynamic and flexible schema. It is mostly used for built-in replication, higher scalability, and availability. There are some advantages to use it over the Standard standalone environment. With the standard Mongo DB, developers have to work on maintenance issues, disk space, set up environments and fix potential problems. Using Mongo DB drivers in Azure Cosmos as a Daas (Database-as-a-service), it solves all the compatibility and reliability issues without any significant concerns providing standard and stable server working behavior environment. Using Daas means developers have the flexibility to grow their project and distribute it globally very efficiently. And since MongoDB drivers in Daas provide most of the significant functionalities like the standard Mongo DB environment, all the features are readily available to use.

For payments, I have used ***Stripe payment*** API. It is a popular gateway for payment processing. It shares the market with PayPal, but unlike PayPal, it is continuously evolving and improving machinery. Some of its clients include Facebook, UNICEF, Lyft and Kickstart. With Stripe you can accept credit cards directly from someone. Without Stripe, any website needs to have a payment processor and payment gateway. This is done through a company which is affiliated with a large bank. It comes with some constraints like contracts. Unlike this stripe cuts all those overheads providing direct ad-hoc services with low charges for each transaction. Stripe is very developer friendly. It is a very robust platform which is compatible with all kinds of online businesses which includes online payments. It has a straightforward API integration with the hosts of other third-party applications. Stripe allows the user to create custom SQL queries for report generations. So basically, it provides a Blackbox entity on both sides by making the transaction look very similar just like any other normal transaction without revealing the stripe connections on the other side. Stripe API is a REST-based API. For internal API errors, it uses HTTP response feature codes to address it.

# CHAPTER 5

PROJECT IMPLEMENTATION

This chapter highlights critical aspects of the implementation of the PodWeb application, showing and explaining screenshots of the final product along with a useful section of code. All the functionality is covered in detail, emphasizing the interaction between the controller and other components.



# **Sign/up and Login**

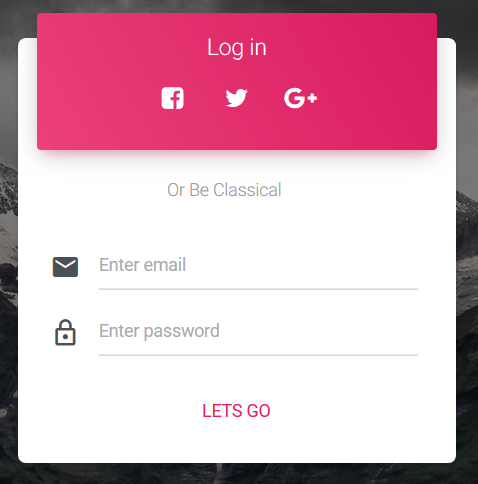
****

Figure 5. 1: Modal for Login Screen

As shown in Figure 5.1, users enter an email address and password to log into the system. There are a couple of steps performed when a user tries to log in into the PodWeb application. The email address and encrypted password are validated which is stored in the database. The email address and user account address are also confirmed which are stored in the blockchain. If both the validation checks are successful, along with a success response to the client, we send JSON web tokens, which will be useful to validate each subsequent user request.

When a user signs up for the PodWeb application, a user account address is created with the use of web3.js API call. There are two ways in which user account is generated using web3.js library version 1.0.x or greater. One way to create user account address is to call web3.eth.accounts.create() and web3.eth.personal.newAccount(password or unique key). Both the methods create new accounts on the blockchain. However, if we use create() method it returns the private key for the newly created account and returning the private key over the connection isn’t secure. We should use create() method to create an account if we are using a local node and store the key locally. If we connect to the remote provider of the blockchain network, then using newAccount() method is more feasible as no key will be generated, and only user account address will be returned which could be associated with the email address.

This user account address generated is stored along with email address in the blockchain with the help of the smart contract in which we use the mapping data type in the solidity programming language. The encrypted password is stored in the database along with email address and user account address. This way two-layer of validation is performed to authenticate the user to use the application. The smart contract snippet used to store the user account address and email address mapping is shown in figure 5.2

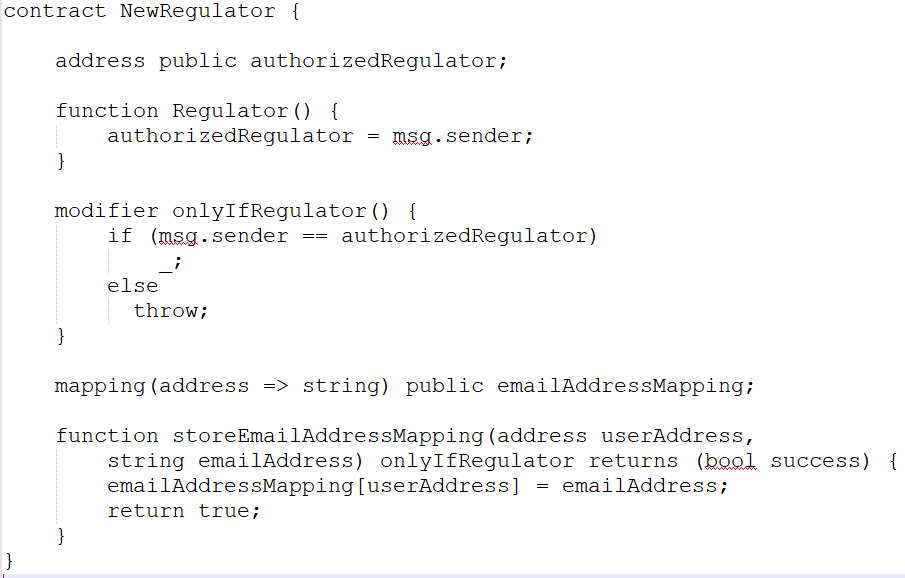


Figure 5. 2: Smart Contract to Store User Account and Email

Address Mapping

The structure of the document for storing user detail metadata in the mongo collection is shown in figure 5.3

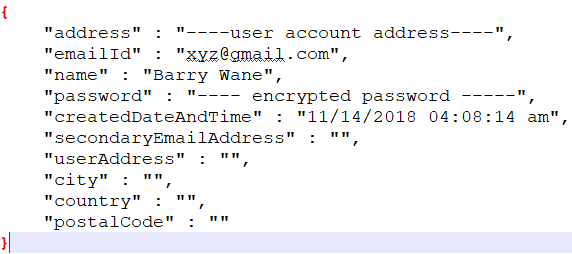


Figure 5. 3: Structure of User Collection in Mongo DB

# **Procedure to Upload Podcast File**

As shown in figure 5.4, the user could upload the podcast by filling the form and selecting the podcast file. The publisher could enter, podcast title, artist name, tags, the choice for either free or paid podcast and amount for the paid podcast and podcast file which will be in mp3 file format. Tags should be selected by publisher in such a way that consumer could easily search through the list of records. One could really promote the podcast by intelligently selecting tags. For example, it is a high possibility for the users to listen to the local, state or national news. Given a proper classification while uploading a podcast using tags, a publisher of the podcast could earn a significant number of tokens if the information is legitimate.

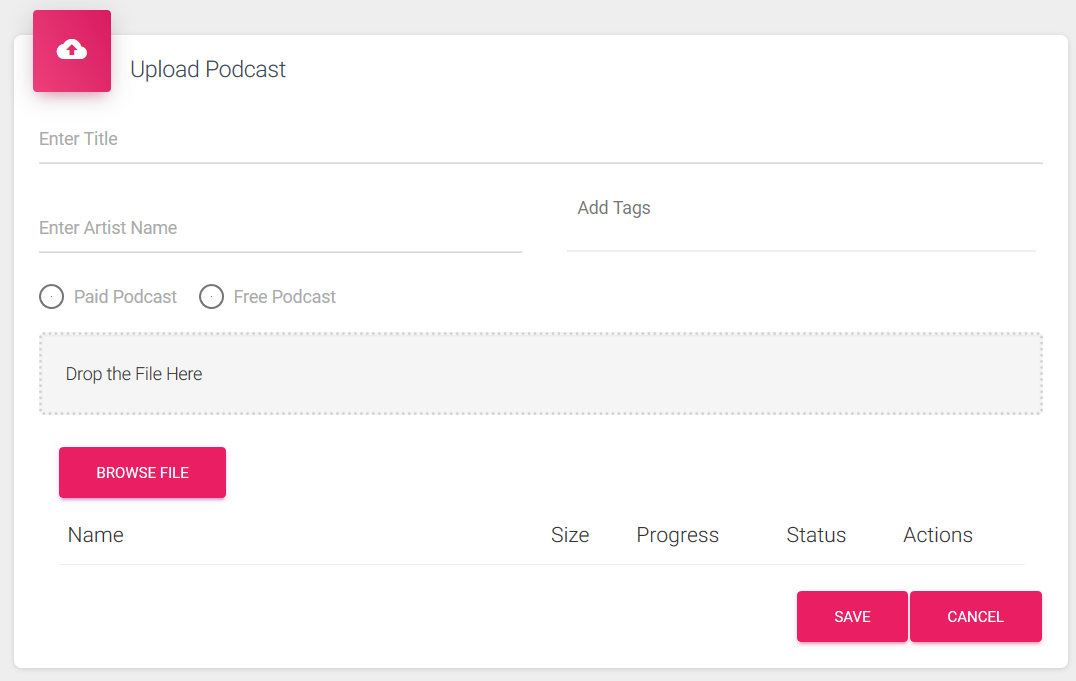


Figure 5. 4: Upload Podcast File

From the client side, form data is uploaded along with file using the ng2-file-upload module. On the server side, Multer library is used which receives the uploaded file and checks if file-type is mp3 or not. If the file is mp3, the file is uploaded to IPFS and verified if that file is already being part of the PodWeb application. So, a file with the same content cannot be uploaded twice, and publisher will be notified that a similar file is already uploaded by some other or oneself, so cannot upload the same podcast file. Given that validation check is successful, the file is uploaded to IPFS, and it returns file hash which is stored in the blockchain along with amount associated for that specific podcast.



Figure 5. 5: Structure of Podcast Detail Collection in Mongo DB

The smart contract used to store the file hash value along with the user account address is shown in figure 5.6. I have created a structure to store an individual file object data against user account address mapping in the smart contract code shown in figure 5.6. As I have used mongo DB instance of Azure cosmos DB, the structure of the document in the mongo collection is shown in figure 5.5

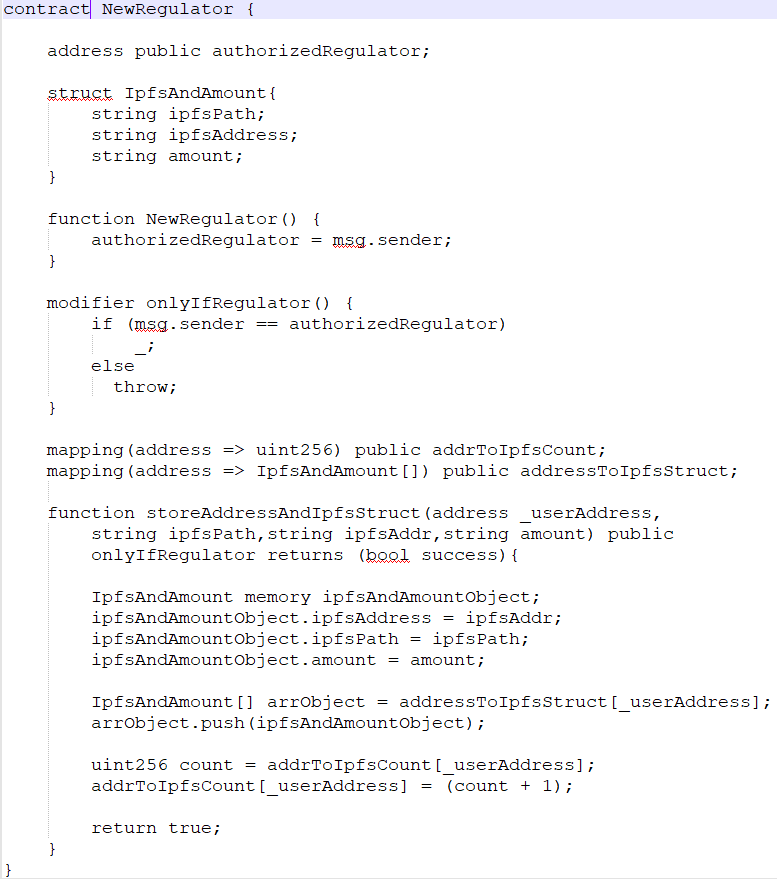


Figure 5. 6: Mapping of File Hash Value and User Account Address

# **Procedure to Publish Podcast File**

Publisher of the podcast has the ability to view a list of all the published podcast by oneself in a data table. They could search and sort the uploaded podcast in the data table. Each podcast’s metadata could be updated by the publisher. They could not update cost and file uploaded for a specific podcast. All the other metadata could be updated, such as podcast title, tags and artist name. Publisher of the podcast could also view which consumer purchased a specific podcast. The name and email address of the consumer are displayed. They could also play the podcast uploaded by oneself. UI, where the list of all published podcast is displayed, is shown in figure 5.7

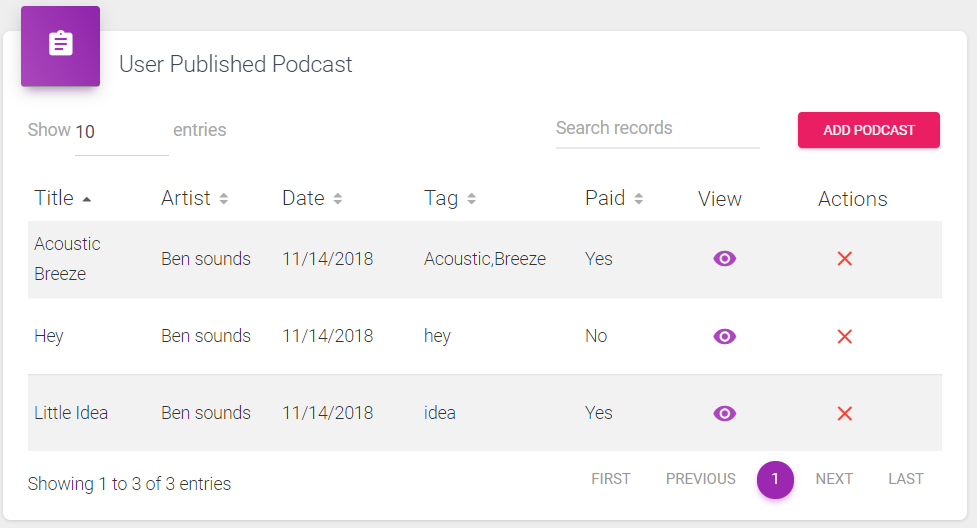
****

Figure 5. 7: Display List of Published Podcasts

This data table is filled by fetching the metadata from Azure cosmos DB as well as from the blockchain. It is a standalone component in AngularJS which loads all the properties for the data-table and populates the data. When component load, a request is placed on a server which invokes ‘getUserPublishedPodcast()’ method to serve its purpose of fetching the data. On the server, the user is validated using JSON web token which is passed as the header. If validation for the user is successful, details from ‘PodcastDetail’ collection is fetched for the validated user. If any of the podcast details are present in the database collection, then all the data is returned with the success message. If no podcast details are found for the validated user, then empty array is passed which signifies no podcast has been uploaded by the user. In that case, data-table will be empty as no records were fetched for the validated user in the database collection used to store podcast details. Publishers who want to update podcast detail, make a service call ‘updatePodcastDetails()’ to the server along with the updated parameters. The server validates user using JSON web tokens and podcast details are updated. As shown in figure 5.8, the publisher could know which consumer has purchased a podcast and could send him promotional emails by knowing consumer’s email address.

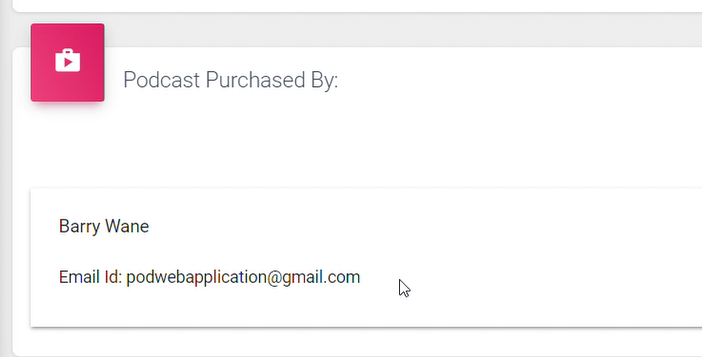


Figure 5. 8: Consumer of a Specific Podcast

# **Procedure to Purchase Tokens**

As per the scope of the application, purchasing some tokens allow users to purchase a paid podcast. However, in the real-world application, the user could transfer the tokens irrespective of purchasing a podcast. Users of the application could select a number of tokens one wants to purchase. Based on the number of tokens selected, the amount in dollars will be populated. The conversion formula from selecting numbers of tokens to amount in dollars is explained in chapter 3. Here, we will discuss what is the workflow for purchasing the tokens, and what is the source of the tokens.

We have deployed a smart contract to generate ERC20 tokens. This is deployed by the administrator. There are 2 smart contracts used in this application for ERC20 tokens,

* PodsToken.sol
* PodsTokenSale.sol

PodsToken.sol, is the smart contract, which defines the name, symbol and the total supply of the tokens when we deploy the contract. This contract contains different mapping and methods which we will go through one by one in detail, but in general, this smart contract contains key components which keep track of all the token details available on blockchain for all the users who are registered for PodWeb application. The name of the ERC20 token deployed on the blockchain is “Pods Token,” a symbol used is “Pods” and total supply is 1,000,000. This total supply is the balance of the administrator who deployed the contract.

Events are the part of solidity code which is used to either send the values in return of the execution of the event or triggers asynchronously to automate the process without user intervention. For my smart contract, I have an event to transfer the tokens from admin to user for joining bonus or for purchasing some tokens. We would like to store tokens owned by the respective user, and we store it using mapping. Tokens owned is mapped to the user account address. So, whenever user joins, a transfer function of the smart contract is invoked from the controller. This transfer function validates if a number of tokens owned by the user from whom tokens will be deducted is greater or equal to the amount of tokens user is requesting to transfer.

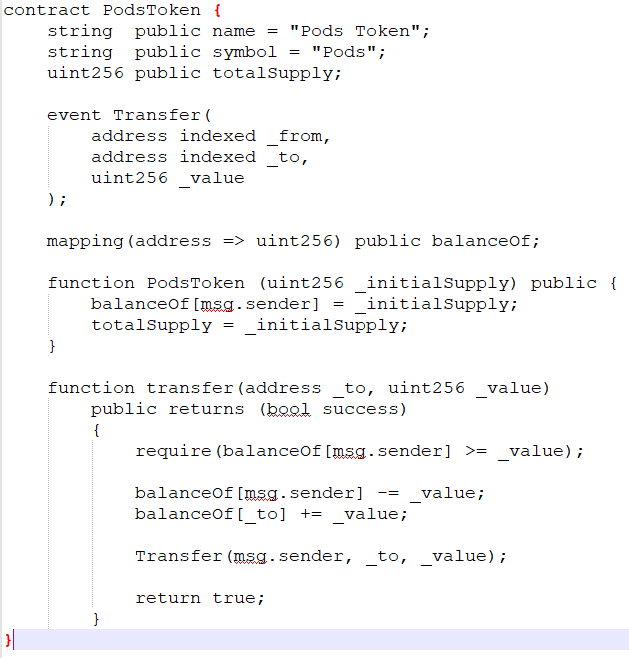


Figure 5. 9: Sample Snippet of PodsToken Smart Contract

If validation check passes, then the amount will be deducted from the user who wants to send the tokens and incremented in the receiver's account which could be tracked using user account address. After that, the event will be triggered which will be validated, and the transaction will be successfully added to the block which will be added to the blockchain. Some of the important snippets are shown for this smart contract is shown in figure 5.9. In smart contract, msg.sender keyword is used for the user account who deployed the contract on the blockchain.

Another smart contract used for generating a sale for the token created in the PodsToken smart contract. This contract imports PodsToken smart contract and keeps tracks of the number of tokens sold at a given instance in the blockchain. It also defines token price which is passed as the arguments along with contract address of PodsToken.sol when we deploy to the blockchain network. Whenever, user purchases a token, ‘buyToken’ function is invoked from this smart contract. It validates a couple of scenarios, before emitting an event to the transfer token function in PodsToken smart contract.

It validates if the product of a number of tokens and token price equals the value which user sent to buy some tokens, it also validates if admin who deployed the contract has sufficient tokens to transfer to the user requesting to purchase some tokens and it increments the token sold count as per the number of tokens transferred. Some of the essential snippets are shown for this smart contract is shown in figure 5.10.

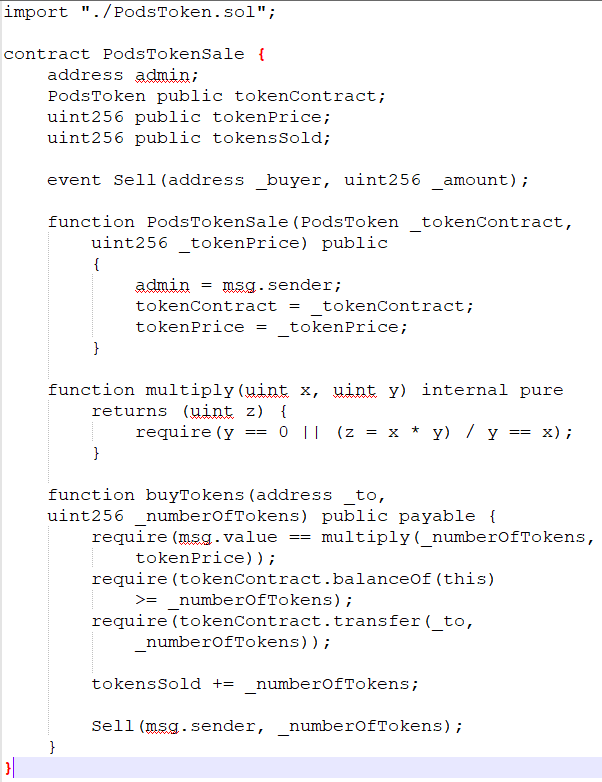


Figure 5. 10: Sample Snippet of PodsTokenSale Smart Contract

From the user’s perspective, we display blockchain details on the client UI. These details are shown on the purchase token component. It is called as token history which keeps track of all the transaction initiated in the blockchain regarding the tokens. It helps users to track down where tokens were used, or which specific podcast was purchased and when. Another mongo DB collection is used whose structure is shown in figure 5.11 to store metadata of the transaction which executed in while tokens are transferred or received.



Figure 5. 11: Structure of Token Purchase Detail Collection

# **Procedure to Display Latest Podcast**

All the users who register for the PodWeb application could view the latest podcast. The latest podcast is the podcast which is published recently by oneself or by other publishers. Latest podcast component is loaded when the user logs into the system. This component contains the data-table which gets populated by making a service call to the backend server. User also has an ability to search a specific podcast. Search is performed on the client side, and it is based on keywords. So, if any keyword is entered in the search bar, results will be filtered based on word present in all the records of the data table. As shown in figure 5.12, the latest podcast is sorted by date when fetched from the database, so it will always bring the most recent records from the database and blockchain.

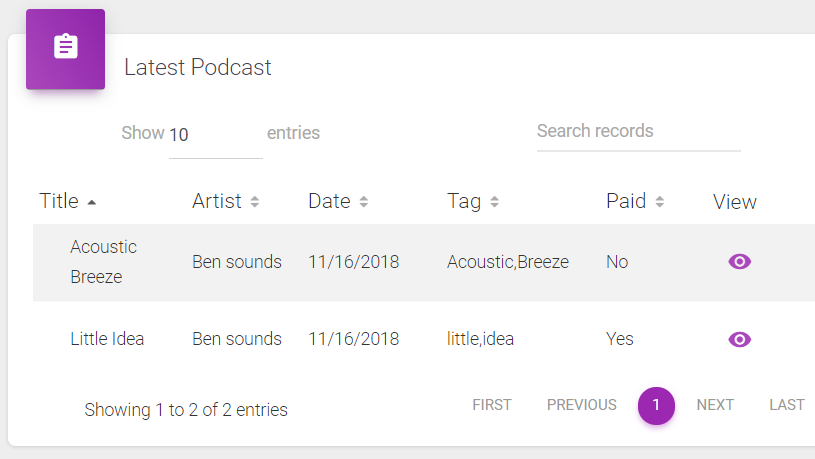


Figure 5. 12: Display List of Latest Podcast

Whenever the latest podcast component is loads, ‘getLatestPodcast’ is the service call which is made on the server. This method, on receiving a POST request, validates the user via headers parameters. On successful validation of user, metadata is fetched from the Mongo DB collection named ‘PodcastDetail.’ For each record fetched from Mongo DB, corresponding file hash value is brought from the blockchain and values are encrypted and passed as the response to the client. On the client side, data-table gets populated with data such as title name, artist name, date, tags, the status of the podcast if it is paid or free and view a specific podcast for more details. The encrypted file hash value is stored as the hidden field which is passed in URL when user select view button for a particular podcast and another service call is made to the server which fetches rest of the metadata for the file.

This view button loads another component. This component is named as Podcast detail component as shown in figure 5.13. It allows the user to purchase the podcast if it is a paid podcast or it allows the user to like and play specific podcast. When this component loads, ‘getPodcastForCurrUser’ service call is made to the server which will validate the user. On successful validation of user, it validates if the path for the encrypted file value is present on the blockchain or not. If the path is found, it validates if the user has already purchased this specific podcast he is requesting for or not. If user has not purchased the podcast, the appropriate response is sent, and the user will be prompted to buy a podcast as shown in figure 5.14. When the user selects, confirms to purchase podcast, the transaction will be placed on blockchain after validating the number of tokens owned by the consumer of the podcast who is trying to purchase. Once, the transaction is validated, the user will be able to access and listen to the podcast from latest as well as purchased podcast data-table.

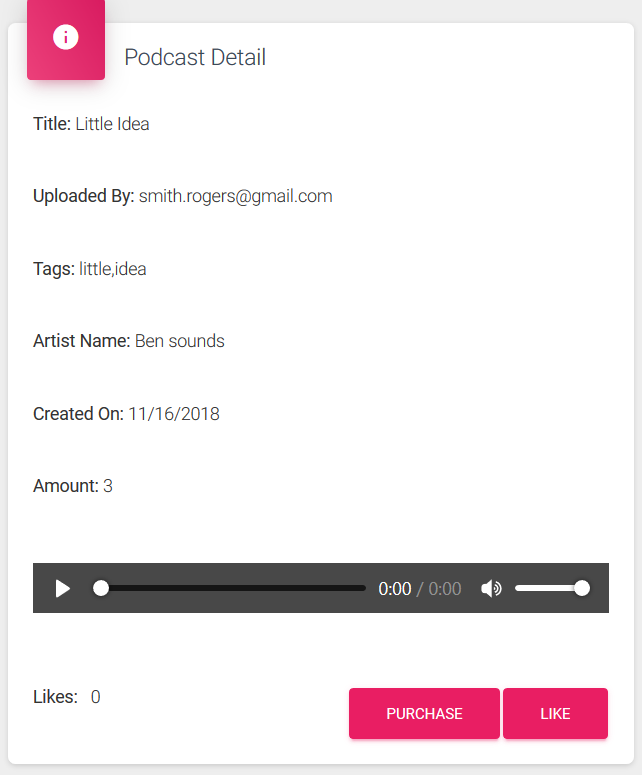


Figure 5. 13: Podcast Detail Component

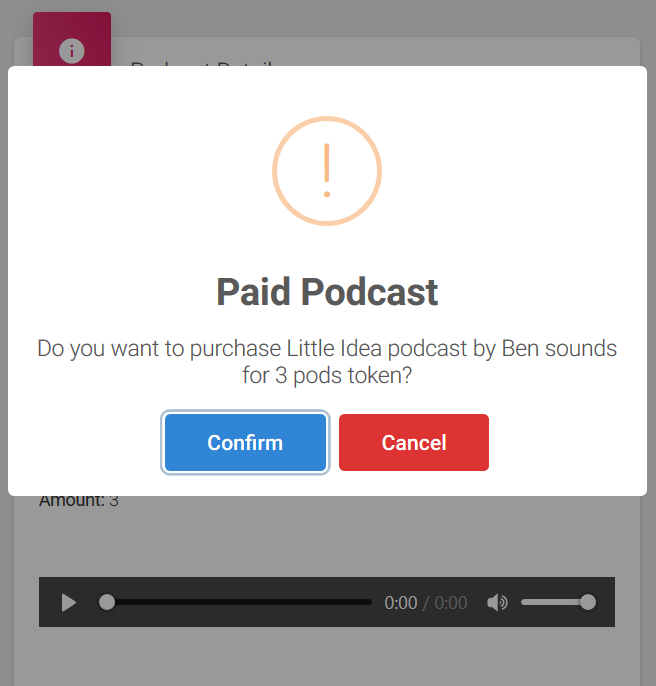
****

Figure 5. 14: Confirm Purchase from Consumer

# **Procedure to Purchase Podcast**

In this component, the user could view a list of all the purchased podcast. It uses data-table to display the list. Like the latest podcast, consumers for a specific podcast could see details from the view button associated with each row in the data table. This reuses the podcast detail component as shown in figure 5.13, by providing the user the flexibility to have access to the podcast file to play on any platform. Loading the reusable component have the same functionality as described in the latest podcast section of this chapter. The component on the client UI looks like as shown in figure 5. 15

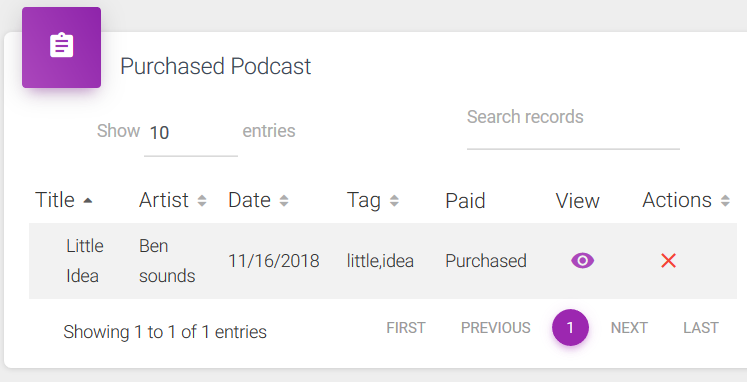


Figure 5. 15: Display List of Purchased Podcast

When purchased podcast list loads, ‘getPurchasedPodcastList’ service call is made to the server. User validation is performed, and if validation check passes, data is fetched from Mongo DB collection named ‘PodcastDetail,’ and corresponding file value is fetched from the blockchain. When the client receives the response, it populates the data with all the required columns and rows.

# 

# CHAPTER 6

PROJECT CONCLUSION

Each day, the popularity of blockchain technology is increasing drastically. People are exploring more and more use-cases because of the nature of the blockchain system. Today, all the major companies like SAP, IBM, Facebook, Bank of America and many more are exploring blockchain technology to increase their business. With the world becoming more digital day-by-day, the use-case implemented in this project will allow users to earn more profit. This project was performed as the simple prototype to move the market of the podcast, music files and other files towards the place where no third-party platform and charges are incurred to make progress in this competitive industry. As, the last chapter of this report, I will discuss what are my learnings and what could be the future scope of this application.



# **Learnings**

Before starting this project, I was only aware of necessary details about Bitcoin and Node JS programming language. This project gave me an opportunity to explore my knowledge and learn what blockchain technology is? How does blockchain work? What are the protocols different blockchain using? What is the transaction? What is the role of miners? What are wallets? What data structure is used to store the information on the blockchain? Finding answers for all this question, which are mentioned in previous chapters, I also enhanced my learning skills by understanding the minute details of the blockchain system. From the implementation perspective, I learned how to create application in Angular JS, how to integrate and use Azure Cosmos DB, how to integrate Stripe Payment API, how to connect to the blockchain network, which library should be used, how to integrate IPFS API, what validation checks would be mandatory to validate the data before performing the transaction and many more things.

# **Future Work**

* One could provide cross-platform support for this application by developing an Android/iOS application
* Allowing files other than mp3 format to upload to the IPFS
* Integrate Apple/Google/Samsung pay to enable the user to have the flexibility to pay the payments for the tokens purchased.
* Sending the notification to the publisher of the podcast, for the purchased podcast by the consumer.
* Role-based access control for publisher and consumer of the podcast.

# BIBLIOGRAPHY

[1] Wondimu Zegeye, (2016 October). WiFi RSS fingerprinting indoor localization for mobile devices [Online]. Available: <https://www.researchgate.net/publication/311609222_WiFi_RSS_fingerprinting_indoor_localization_for_mobile_devices>

[2] Minh Tu Hoang. (2019 March). Recurrent Neural Networks for Accurate RSSI Indoor Localization? [Online] Available: <https://arxiv.org/pdf/1903.11703.pdf>

[3] Investopedia. Proof of stake (PoS) [Online]

Available: <https://www.investopedia.com/terms/p/proof-stake-pos.asp>

[4] EtherInc (2018 May). EtherInc (ETI) ICO Review – ICO Token News [Online] Available: <https://www.icotokennews.com/icos/etherinc/>

[5] Solidity. [Online]. Available: <https://solidity.readthedocs.io/en/develop/>

[6] Satoshi Nakamoto. (2008, Oct). Bitcoin: A Peer-to-Peer Electronic Cash System. [Online]. Available: <https://bitcoin.org/bitcoin.pdf>

[7] Vitalik Buterin. (2013). Ethereum Whitepaper. [Online]. Available:

<https://github.com/ethereum/wiki/wiki/White-Paper>

[8] AngularJS Icon [Online] Available: <https://angular.io/presskit>

[9] Azure Cosmos DB Icon [Online] Available: <https://azure.microsoft.com/en-us/services/cosmos-db/>

[10] Stripe Icon [Online] Available: <https://stripe.com/newsroom/brand-assets>

[11] Ethereum Icon [Online] Available: <https://www.ethereum.org/>

[12] Anders Brownworth. Blockchain Demo [Online] Available: <https://anders.com/blockchain/>

[13] Brandon Morelli, 25 Node JS Tutorials (2018) [Online] Available: <https://codeburst.io/25-node-js-tutorials-1db3b1da0260>

[14] Stripe Docs [Online] Available: <https://stripe.com/docs/checkout/express>

[15] Stephen Grider, Ethereum and Solidity: The Complete Developer's Guide [Online] Available: <https://www.udemy.com/ethereum-and-solidity-the-complete-developers-guide/>

[16] Azure Cosmos DB Documentation [Online] Available: <https://docs.microsoft.com/en-us/azure/cosmos-db/>

[17] Guide to AngularJS Documentation [Online] Available: <https://code.angularjs.org/snapshot/docs/guide>